# Transparent Data Encryption (TDE) in Oracle 10g Database Release 2

Oracle has many security features available within the database, but until now there has been no "out-of-the-box" method for protecting the data at the operating system level. The Transparent Data Encryption (TDE) feature introduced in Oracle 10g Database Release 2 allows sensitive data to be encrypted within the datafiles to prevent access to it from the operating system. This article presents some basic examples of its use.

Related articles.

- Tablespace Encryption in Oracle 11g Database Release 1
- Data Encryption - DBMS_OBFUSCATION_TOOLKIT
- Database Security Enhancements in Oracle Database 10g - DBMS_CRYPTO
- SecureFiles in Oracle 11g Database Release 1 - LOB Encryption

## Setup

In order to show the encryption working we need to open a datafile in a HEX editor. Rather than trying to open a huge datafile, it makes sense to create a small file for this test.

```
CONN sys/password AS SYSDBA
```

```
CREATE TABLESPACE tde_test
  DATAFILE '/u01/oradata/DB10G/tde_test.dbf' SIZE 128K
  AUTOEXTEND ON NEXT 64K;
```

Next, create a user with with a quota on the new tablespace.

```
CREATE USER test IDENTIFIED BY test DEFAULT TABLESPACE tde_test;
ALTER USER test QUOTA UNLIMITED ON tde_test;
GRANT CONNECT TO test;
GRANT CREATE TABLE TO test;
```

## Normal Column

First we will prove that the data from a normal column can be seen from the OS. To do this create a test table and insert some data.

```
CONN test/test

CREATE TABLE tde_test (
  id    NUMBER(10),
  data  VARCHAR2(50)
)
TABLESPACE tde_test;

INSERT INTO tde_test (id, data) VALUES (1, 'This is a secret!');
COMMIT;
```

Then flush the buffer cache to make sure the data is written to the datafile.

```
CONN sys/password AS SYSDBA
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

Open the datafile using a HEX editor (like UltraEdit) and the sentence "This is a secret!" is clearly visible amongst all the non-printable characters.

## Encrypted Column

Before attempting to create a table with encrypted columns, a wallet must be created to hold the encryption key. The search order for finding the wallet is as follows:

1. If present, the location specified by the `ENCRYPTION_WALLET_LOCATION` parameter in the sqlnet.ora file.
2. If present, the location specified by the `WALLET_LOCATION` parameter in the sqlnet.ora file.
3. The default location for the wallet ($ORACLE_BASE/admin/$ORACLE_SID/wallet).

Although encrypted tablespaces can share the default database wallet, Oracle recommend you use a separate wallet for transparent data encryption functionality by specifying the `ENCRYPTION_WALLET_LOCATION` parameter in the sqlnet.ora file. To accomplish this we add the following entry into the sqlnet.ora file on the server and make sure the specified directory has been created.

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
    (DIRECTORY=/u01/app/oracle/admin/DB10G/encryption_wallet/)))
```

The following command creates and opens the wallet.

```
CONN sys/password AS SYSDBA
-- 10g version
ALTER SYSTEM SET ENCRYPTION KEY AUTHENTICATED BY "myPassword";

-- 11g version
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "myPassword";
```

Wallets must be reopened after an instance restart and can be closed to prevent access to encrypted columns.

```
-- 10g version
ALTER SYSTEM SET ENCRYPTION WALLET OPEN AUTHENTICATED BY "myPassword";

-- 11g version
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "myPassword";

ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
```

Create a test table with an encrypted column and insert some data. Using the `ENCRYPT` clause on its own is the same as using the `ENCRYPT USING 'AES192'` clause, as AES192 is the default encryption method.

```
CONN test/test

DROP TABLE tde_test;
PURGE RECYCLEBIN;
```

```
CREATE TABLE tde_test (
  id    NUMBER(10),
  data  VARCHAR2(50) ENCRYPT
)
TABLESPACE tde_test;

INSERT INTO tde_test (id, data) VALUES (1, 'This is a secret!');
COMMIT;
```

Flush the buffer cache to make sure the data is written to the datafile.

```
CONN sys/password AS SYSDBA
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

When the file is opened using a HEX editor only non-printable characters are present. The test sentence cannot be seen anywhere, but the data is still clearly visible from a database connection.

```
SELECT * FROM tde_test;

        ID DATA
---------- --------------------------------------------------
         1 This is a secret!

1 row selected.
```

## Database Startup

The following text shows the impact of stopping and starting the database on an encrypted column. Notice how the wallet must be opened before the data is accessible.

```
SQL> CONN / AS SYSDBA
Connected.
SQL> SHUTDOWN IMMEDIATE;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP
```

```
ORACLE instance started.

Total System Global Area  626327552 bytes
Fixed Size                  2255832 bytes
Variable Size             234882088 bytes
Database Buffers          381681664 bytes
Redo Buffers                7507968 bytes
Database mounted.
Database opened.
SQL> SELECT tablespace_name, encrypted, status FROM dba_tablespaces;

TABLESPACE_NAME                ENC STATUS
------------------------------ --- ---------
SYSTEM                         NO  ONLINE
SYSAUX                         NO  ONLINE
UNDOTBS1                       NO  ONLINE
TEMP                           NO  ONLINE
USERS                          NO  ONLINE
EXAMPLE                        NO  ONLINE
SOE                            NO  ONLINE
TDE_TEST                       NO  ONLINE

8 rows selected.

SQL> SELECT * FROM test.tde_test;
SELECT * FROM test.tde_test
                   *
ERROR at line 1:
ORA-28365: wallet is not open


SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "myPassword";

System altered.

SQL> SELECT * FROM test.tde_test;

        ID DATA
---------- --------------------------------------------------
         1 This is a secret!

1 row selected.
```

```
SQL>
```

## Performance

There is a performance overhead associated with the encryption/decryption process. The following tables are used in a performance comparison.

```
CONN test/test
CREATE TABLE tde_test_1 (
  id    NUMBER(10),
  data  VARCHAR2(50)
)
TABLESPACE tde_test;

CREATE TABLE tde_test_2 (
  id    NUMBER(10),
  data  VARCHAR2(50) ENCRYPT
)
TABLESPACE tde_test;
```

The following script uses these tables to compare the speed of regular and encrypted inserts and regular and decrypted queries. Each test repeats 1000 times, with the timings reported in 100ths of a second.

```
SET SERVEROUTPUT ON SIZE UNLIMITED
DECLARE
  l_loops  NUMBER := 1000;
  l_data   VARCHAR2(50);
  l_start  NUMBER;
BEGIN
  EXECUTE IMMEDIATE 'TRUNCATE TABLE tde_test_1';
  EXECUTE IMMEDIATE 'TRUNCATE TABLE tde_test_2';

  l_start := DBMS_UTILITY.get_time;
  FOR i IN 1 .. l_loops LOOP
    INSERT INTO tde_test_1 (id, data)
    VALUES (i, 'Data for ' || i);
  END LOOP;
  DBMS_OUTPUT.put_line('Normal Insert   : ' || (DBMS_UTILITY.get_time - l_start));
```

Translate

```
  l_start := DBMS_UTILITY.get_time;
  FOR i IN 1 .. l_loops LOOP
    INSERT INTO tde_test_2 (id, data)
    VALUES (i, 'Data for ' || i);
  END LOOP;
  DBMS_OUTPUT.put_line('Encrypted Insert: ' || (DBMS_UTILITY.get_time - l_start));

  l_start := DBMS_UTILITY.get_time;
  FOR i IN 1 .. l_loops LOOP
    SELECT data
    INTO   l_data
    FROM   tde_test_1
    WHERE  id = i;
  END LOOP;
  DBMS_OUTPUT.put_line('Normal Query    : ' || (DBMS_UTILITY.get_time - l_start));

  l_start := DBMS_UTILITY.get_time;
  FOR i IN 1 .. l_loops LOOP
    SELECT data
    INTO   l_data
    FROM   tde_test_2
    WHERE  id = i;
  END LOOP;
  DBMS_OUTPUT.put_line('Decrypted Query : ' || (DBMS_UTILITY.get_time - l_start));
END;
/
Normal Insert   : 31
Encrypted Insert: 45
Normal Query    : 42
Decrypted Query : 58

PL/SQL procedure successfully completed.

SQL>
```

The results clearly demonstrate that encrypted inserts and decrypted queries are slower than their normal counterparts.

## External Tables

External tables can be encrypted in a similar way to regular tables. First, we make sure the default data pump directory is available to the test user.

```
CONN sys/password AS SYSDBA
GRANT READ, WRITE ON DIRECTORY data_pump_dir TO test;
```

Next, we create the external table as a copy of an existing table, using the `ENCRYPT` clause.

```
CONN test/test

CREATE TABLE tde_test_1_ext (
  id,
  data ENCRYPT IDENTIFIED BY "myPassword"
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY data_pump_dir
  location ('tde_test_1_ext.dmp')
)
AS
SELECT id,
       data
FROM   tde_test_1;
```

## Views

The `%_ENCRYPTED_COLUMNS` views are used to display information about encrypted columns.

```
SET LINESIZE 100
COLUMN owner FORMAT A15
COLUMN tble_name FORMAT A15
COLUMN column_name FORMAT A15

SELECT * FROM dba_encrypted_columns;

OWNER           TABLE_NAME                     COLUMN_NAME     ENCRYPTION_ALG                SAL
--------------- ------------------------------ --------------- ----------------------------- ---
TEST            TDE_TEST_2                     DATA            AES 192 bits key              YES
TEST            TDE_TEST_1_EXT                 DATA            AES 192 bits key              YES

2 rows selected.
```

```
SQL>
```

## Miscellaneous Information

- The master key can be reset in a similar way to how it is set initially. The instructions for this are shown here.
- The wallet password can be changed using the Oracle Wallet Manager. The instructions for this are shown here.
- The encryption algorithm used for a column can be changed using the `REKEY` command. The instructions for this are shown here.

For more information see:

- Transparent Data Encryption
- Tablespace Encryption in Oracle 11g Database Release 1
- Data Encryption - DBMS_OBFUSCATION_TOOLKIT
- Database Security Enhancements in Oracle Database 10g - DBMS_CRYPTO
- SecureFiles in Oracle 11g Database Release 1 - LOB Encryption
- Managing TDE wallets in a RAC environment [ID 567287.1]

Hope this helps. Regards Tim...

Back to the Top.

6 comments, read/add them...

g+1 3 Gosto 5 Tweet 0 6